Summary of "Amorized Efficiency" - by Cody Averett

The article discusses the overall complexity (amortized) of the "move to front" and "least recently used" algorthims. They set out to prove these two alorithms to be superior to the alternative rules.

The first example compares the performance of the "move to front" (MF), "transpose" (T), and "frequency count" (FC) algorithms in regards to how they impact access, insert, and delete operations performed upon a "self-organizing" dictionary.

Afterwards they discussed an example of paging where they introduce several related algorithmic rules that may potentially be used when working with a pages of a fixed size. The problem depicts reading data from both fast and slow memory. Reading values from fast memory incures no performance cost whereas any read to slow memory would first need to be moved into fast memory. They mentioned the "longest forward distance" to be the most equivelant to "move to front" in regards to insert operations.

The culmination of this study suggests, in this case, that overall "move to front" esque operations are often the best performance-wise to use due to it having equal or better performance to the next best competing algorithms.

I found this article to be fairly complex even though I come from what I would descibe as a technical background. It was difficult to follow through some of the examples initially due to the articles heavy use of initialisms and notations I had not yet been aquainted with. What I took away from reviewing this paper is that it is important to measure the overall performance of an algorithm's upper bound, the amortized results, and compare said results to the optimum algorithm in order to discern which algorithms prove to give the overall best real world performance.